Taller 7: Grafos

Objetivos

- Estudiar, implementar, probar y documentar la estructura de datos grafo no dirigido
- Utilizar adecuadamente herramientas para el desarrollo de software en equipos

Fechas Límite de Entrega

Diseño: 7 de mayo, 11:59 p.m.

Lectura Previa

Estudiar la teoría de los grafos dirigidos con peso. Consultar las secciones 4.1 a 4.4 del libro guía *Algorithms* de Sedgewick y Wayne.

Fuentes de Datos

A continuación, se presenta una descripción de las fuentes de datos que se utilizarán en el taller.

Estaciones de Policía

La información de las estaciones de policía se debe construir a partir de la carga del archivo *estacionpolicia.geojson* (formato JSON) que se pueden descargar del portal Web de consulta de datos abiertos del sistema PRUDENCIA de Bogotá en el enlace: https://datosabiertos.bogota.gov.co/dataset/estacion-de-policia-para-bogota

El archivo de estaciones de policia tiene la siguiente estructura:

```
{
	"type":"FeatureCollection",
	"crs":
	{
		"type":"name",
		"properties":
		{
			"name":"EPSG:4686"
		}
	},
```

```
"features":[
{
	"type":"Feature",
	"id":1637,
	"geometry":{
		"type":"Point",
		"coordinates":[
			-74.103158699999938,
			4.5856918000000633
		]
	},
	"properties":
	{
		"OBJECTID":1637,
		"EPOCOD_PLAN":5,
		"EPOCOD_ENT":"137",
		"EPOCOD_PROY":"7507",
		"EPOANIO_GEO":2016,
		"EPOFECHA_INI":1464739200000,
		"EPOFECHA_FIN":1609372800000,
		"EPODESCRIP":"Estación de Policía Antonio Nariño",
		"EPOEST_PROY":"TERM",
		"EPOINTERV_ESP":"NIES",
		"EPODIR_SITIO":"KR 24 CL 18 - 90 SUR",
		"EPOCOD_SITIO":"020102",
		"EPOLATITUD":4.5856236599999995,
		"EPOLONGITU":-74.103130640000003,
		"EPOSERVICIO":"Tiene como finalidad asegurar y ejercer el control de la
		jurisdicción, prestar un servicio integral de vigilancia urbana y
		rural, desarrollar los procesos de prevención, disuasión, investigación
		y control de los delitos y contravenciones apoyados en la gestión
		territorial de la seguridad ciudadana con las autoridades locales
		político administrativas.",
		"EPOHORARIO":"24 horas",
		"EPOTELEFON":"5521110",
		"EPOCELECTR":"mebog.e15@policia.gov.co",
		"EPOCONTACT":"Policía Nacional",
		"EPOPWEB":"http://www.policia.gov.co",
		"EPOIUUPLAN":"UPZ38",
		"EPOIUSCATA":"002103",
		"EPOIULOCAL":"15",
		"EPOEASOCIA":"No Aplica",
		"EPOFUNCION":"Coercion",
		"EPOTEQUIPA":"Justicia",
		"EPONOMBRE":"Estación de Policía Antonio Nariño",
		"EPOIDENTIF":"EPO003",
		"EPOFECHA_C":1565740800000
	}
}

{	…

 },
```

```
        … ]

}
```

Malla Vial de Bogotá

La malla vial de Bogotá, representada como un grafo donde los vértices son las intersecciones de las calles y los arcos las calles, puede ser utilizada en una gran cantidad de aplicaciones para navegación terrestre. A continuación, se mostrarán unos ejemplos con el contenido de los archivos de *bogota_vertices.txt* y *bogota_arcos.txt*:

Se representarán las **intersecciones de la malla vial** como los **vértices del grafo**. Estas intersecciones están representadas en el archivo *bogota_vertices.txt* con id, longitud, y latitud:

```
0,-74.08921298299998,4.582989396000016
1,-74.08952746199998,4.582560966000017
```

Las vías de la malla vial que conectan las intersecciones están almacenadas en el archivo *bogota_arcos.txt*. Estas vías se representan como un conjunto de adyacencias donde el primer campo especifica el id del vértice origen, y los demás campos son los ids de sus vértices adyacentes:

```
0 1 733
1 49 …
```

Lo que su grupo debe hacer (parejas)

Parte 1 – Configuración del Repositorio

1. Cree en GitHub un repositorio llamado T7_202010.
2. Cree el README del repositorio donde aparezcan los nombres y códigos de los miembros del grupo de trabajo.
3. Realice el procedimiento para crear el directorio en su computador de trabajo para que relacione este directorio con el repositorio remoto que acaba de crear.
4. Descargue los datos descritos en la sección fuentes de datos y cópielos en la carpeta data del repositorio local.

Parte 2 – Desarrollo

5. Implemente la estructura de datos **GrafoNoDirigido** que soporte vértices con llave genérica (*K*). Para este caso se seleccionó un grafo no dirigido dado que no se tendrá en cuenta la dirección de las vías.
6. El API que debe implementar está compuesto, al menos, por las siguientes operaciones:

Operación	Descripción
`Graph (int n)`	Crea un grafo No dirigido de tamaño **n** vértices y sin arcos
`int V()`	Obtener el número de vértices
`int E()`	Obtener el número de arcos. Cada arco No dirigido debe contarse una única vez.
`void addEdge(K idVertexIni, K idVertexFin, double cost)`	Adicionar el arco No dirigido entre el vértice IdVertexIni y el vértice IdVertexFin. El arco tiene el costo *cost*.
`V getInfoVertex(K idVertex)`	Obtener la información de un vértice. Si el vértice no existe retorna null.
`void setInfoVertex(K idVertex, V infoVertex)`	Modificar la información del vértice idVertex
`double getCostArc(K idVertexIni, K idVertexFin)`	Obtener el costo de un arco, si el arco no existe, retorna –1.0
`void setCostArc(K idVertexIni, K idVertexFin, double cost)`	Modificar el costo del arco No dirigido entre los vértices idVertexIni e idVertexFin
`void addVertex(K idVertex, V infoVertex)`	Adiciona un vértice con un Id único. El vértice tiene la información InfoVertex.
`Iterable <K> adj (K idVertex)`	Retorna los identificadores de los vértices adyacentes a idVertex
`void uncheck()`	Desmarca todos los vértices del grafo
`void dfs(K s)`	Ejecuta la búsqueda de profundidad (DFS) sobre el grafo con el vértice s como origen. Los vértices resultado de la búsqueda quedan marcados y deben tener información que pertenecen a una misma componente conectada.
`int cc()`	Obtiene la cantidad de componentes conectados del grafo. Cada vértice debe quedar marcado y debe reconocer a cuál componente conectada pertenece. En caso de que el grafo esté vacío, retorna 0.
`Iterable<K> getCC(K idVertex)`	Obtiene los vértices alcanzados a partir del vértice idVertex después de la ejecución de los metodos dfs(K) y cc().

7. Diseñe escenarios de prueba para comprobar el correcto funcionamiento de la estructura.

8. Construya sets de pruebas (es decir casos de prueba en JUnit) para verificar y validar la implementación del API.

9. Desarrolle un método para cargar el grafo a partir de las fuentes de datos. El objetivo es construir un **Grafo No Dirigido** que represente la malla vial de Bogotá tomando como base la información de los archivos *bogota_vertices.txt y bogota_arcos.txt.*

- En caso de que un id de vértice encontrado dentro del archivo de arcos, no esté en el archivo de vértices, se debe omitir la creación de ese arco.
- El costo de cada arco es la **distancia haversiana** entre sus dos vértices en km. La fórmula de distancia haversiana calcula la distancia en kilómetros entre dos localizaciones geográficas sobre la superficie de la tierra.
- Verifique que el grafo quedó bien creado en memoria y reporte la cantidad de vértices y arcos del grafo por consola.

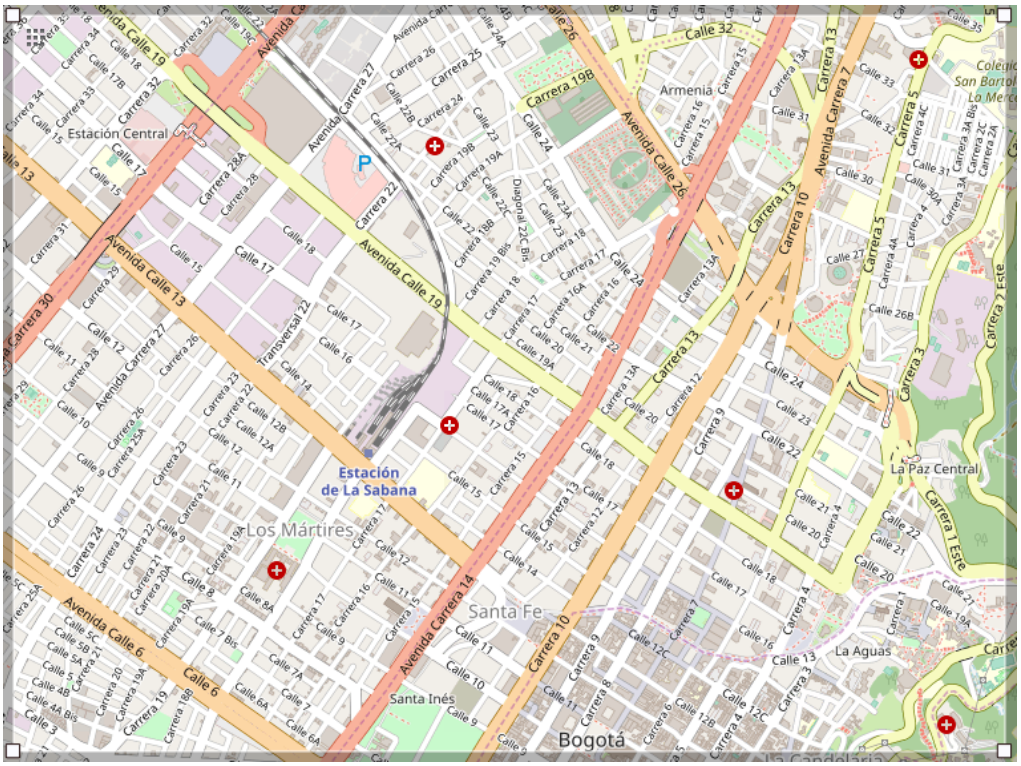
Sugerencia 1: Utilizar un API o código ya disponible para calcular la distancia haversiana entre dos puntos. Por ejemplo: https://github.com/jasonwinn/haversine/blob/master/Haversine.java

Sugerencia 2: En caso de obtener un error por desbordamiento de la pila (Stack overflow error), puede incrementar el tamaño de la pila a 4M u 8M usando el argumento –Xss, p. ej: *-Xss4M*

10. Desarrolle un método para guardar el grafo del punto anterior en un archivo JSON. Defina el esquema JSON para persistir el grafo teniendo en cuenta que para cada vértice se debe almacenar su posición y de cada arco su distancia haversiana. Este grafo extendido con las distancias harvesianas se va a utilizar en el proyecto 3.

11. Desarrolle un método para cargar el grafo persistido en el paso anterior, leyendo el archivo JSON definido por usted, verifique que el grafo quedó bien creado en memoria y reporte la cantidad de vértices y arcos del grafo por consola.

12. Grafique con ayuda de Google Maps la parte del grafo resultante de la carga de la zona delimitada por las siguientes coordenadas: Longitud min= -74.094723, Longitud max= -74.062707, Latitud min= 4.597714 y Latitud max= 4.621360.



Pintar los nodos intersección (con círculos), y los arcos (que conectan los nodos).

Sugerencia 3: Utilizar el API JxMaps (https://www.teamdev.com/jxmaps), mapbox (https://docs.mapbox.com/android/java/overview/#installation) para Java, o Maps JavaScript API (https://developers.google.com/maps/documentation/javascript/tutorial?hl=es).

13. Realizar la carga de datos del archivo *estacionpolicia.geojson*. Revise la estructura del archivo y cargue los datos que considere relevantes. Verifique que el archivo quedó bien creado en

memoria y reporte la cantidad de estaciones por consola y la información cargada para cada estación.

14. Agregue al mapa generado en el punto 12 las estaciones de policía que se encuentren dentro de la zona delimitada; para tal fin haga uso de los datos de localización etiquetados como "EPOLATITUD" y "EPOLONGITU" del archivo *estacionpolicia.geojson*.

Entrega

1. Para hacer la entrega del taller usted debe agregar a su repositorio los usuarios de los monitores y su profesor, siguiendo las instrucciones del documento “Guía Creación de Repositorios para Talleres y Proyectos.docx”.
2. Entregue su taller por medio de GitHub. Recuerde, si su repositorio No tiene acceso al profesor ni a los monitores, su taller no será calificado por más de que lo haya desarrollado.